A Comparison of Large-Scale Overlay Management Techniques

Sushant Jain, Ratul Mahajan, David Wetherall, Gaetano Borriello, and Steven D. Gribble
{*sushjain,ratul,djw,gaetano,gribble*}*@cs.washington.edu*

Computer Science and Engineering
University of Washington
Seattle, WA, 98195-2350

Technical Report UW-CSE 02-02-02

Abstract

In this paper we present Kudos, a novel hierarchical, topology aware overlay construction algorithm. Kudos is an extension of Narada, an existing topology aware overlay; by adding hierarchy, we have significantly increased the scalability of Kudos while maintaining the performance advantage of topology awareness. Additionally, we provide the first detailed comparison between topology aware and topology agnostic overlay construction algorithms, by performing detailed simulations of Kudos, CAN [1], Chord [2], and a "random power law" overlay. We show that all overlay topologies are faced with a fundamental tradeoff between relative delay penalty (RDP) and link stress, and that by changing node out-degree, different points in this tradeoff can be selected. We demonstrate that Kudos significant outperforms all considered topology agnostic algorithms, both in terms of latency and bandwidth, although it cannot scale to the same degree because of the costs incurred by topology aware overlay maintenance.

1 Introduction

Overlays have recently become a popular way to deploy new network protocols and middleware services. An overlay is a virtual network between participating nodes; each pair of nodes in the overlay are separated by one or more links in the underlying physical network topology, and some pairs of overlay nodes are connected by a *tunnel* through the physical network. Because overlay traffic is tunneled, overlay protocols have the advantage of incremental deployability. Early overlays such as the M-Bone [3] and the 6-Bone [4] were intended as transitional solutions, but overlays have since been recognized as an effective, permanent way to deploy new services, including application-level multicast [5, 6] and peer-to-peer networks [1, 2, 7, 8, 9].

Because the topology of an overlay may differ from that of the underlying physical network, the overlay may be inefficient: packets flowing over distinct virtual links may cross the same physical link, and the route of a single packet through the overlay may involve repeated crossings over the same physical link. The algorithm or mechanism used to form an overlay is therefore critical to its performance: overlay topologies that closely match their underlying physical network topology are more efficient, in general. While overlays such as the M-Bone [3] were manually configured and relatively static in topology, recent overlays are built and managed with self-organizing algorithms. These algorithms adapt the overlay network topology to changes in node membership and to changes in the physical network, such as the onset of congestion.

Topology aware algorithms, such as RON [10], Narada [5], and RMX [11], use active measurements to infer network properties and to make an informed choice in overlay topology. *Topology agnostic* overlays, such as CAN [1], Chord [2], Tapestry [8], and Pastry [9] use other criteria to form overlays, such as embedded the overlay in a virtual geometric space to simplify routing. Topology aware algorithms can avoid the inefficiencies of overlays, but do so at the cost of increased management overhead and potentially poor scalability.

This paper makes two contributions: first, it presents the design of Kudos, a hierarchical extension to the Narada topology aware overlay construction algorithm. Because of its use of hierarchy, Kudos has lower management overhead and superior scalability than Narada; furthermore, we show that the advantages gained by hierarchy do not come at the cost of poorer routing efficiency.

Our second contribution is to present a detailed comparison of the advantages and disadvantages that topology aware algorithms have relative to topology agnostic overlays. Using packet-level simulations, we compare Kudos to CAN and Chord, showing that that the routing efficiency of these topology agnostic overlays (as measured by *relative delay penalty*) is at least a factor of two worse than that of Kudos; this is true even using recent heuristic extensions to CAN and Chord that were designed to address these very issues. We further demonstrate that Kudos is better able to achieve lower packet delivery latency and higher bandwidth than both CAN and Chord.

In the next section of the paper, we discuss how to evaluate various overlay topologies. Section 3 describes the topology aware approach, Kudos. Section 4 describes various topology agnostic algorithms that we use for comparison. Section 5 describes the experimental methodology. Results of experiments are presented in Section 6 and Section 7. We talk about related work in Section 9, and we conclude in Section 10.

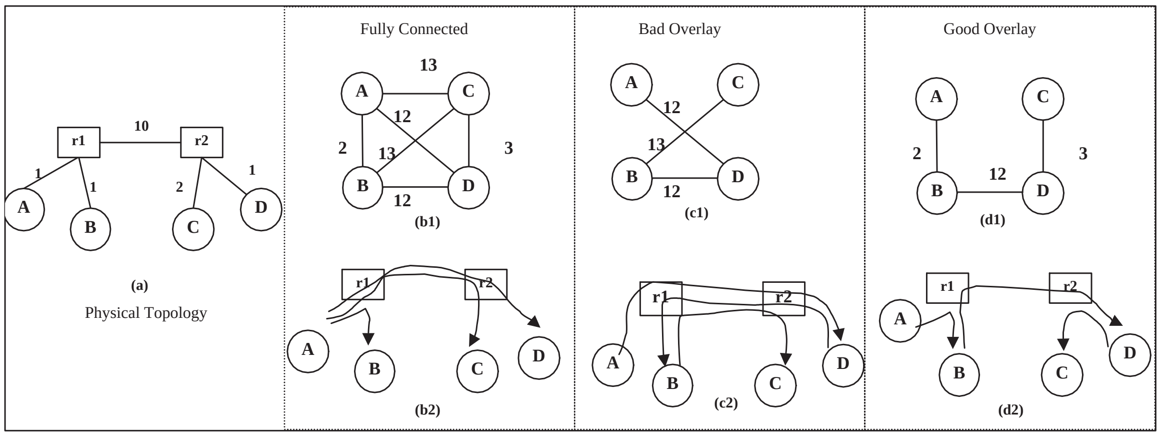

Figure 1: Example illustrating different overlay topologies. (a) shows a physical network with 4 hosts interested in participating in overlay. Three different overlays topologies are shown. Below them is shown how A would broadcast data to B,C,D using the overlay.

2 On the Evaluation of Overlay Topologies

As previously argued, topology aware overlay construction algorithms attempt to increase the efficiency of routing across the overlay, at the cost of increased management overhead. Because topology aware algorithms use periodic probes to measure the characteristics of all end-to-end paths, these algorithms may also suffer from limited scalability, effectively restricting them to group sizes of approximate 100 nodes.

2.1 Overlay Performance Metrics

We use multicast as a representative driving application to quantify the efficiency of an overlay. Several recent proposals have discussed overlay management in the context of application-level multicast [5, 12, 6, 13]; the proposals typically construct multicast distribution trees rooted at the source of a multicast transmission. Regardless of the specific details of each particular proposal, two metrics that can be used to evaluate performance are *relative delay penalty (RDP)* and *stress.*

RDP is a measure of the additional packet delay introduced by overlay on the delivery of a single packet between a source and destination. More specifically, RDP is the ratio of the latency experienced when sending data using the overlay to the latency experienced when sending data directly using the underlying network.

Stress is a measure of the excess bandwidth consumption induced by the overlay during a multicast transmission. The stress of a physical link is defined as the number of overlay tunnels that send traffic over that link. Note that stress is both a function of the topology and the multicast tree used: flooding-style broadcasts cause more stress on a physical link than multicasts. For efficient multicast trees, a multicast packet flows over each virtual overlay tunnel at most once.

Ideally, an overlay should have both low RDP and low stress. Unfortunately, these requirements can conflicting. To see this, consider Figure 1(a), which shows an example physical network with four hosts interested in forming an overlay. Figure 1(b1) shows a fully connected overlay topology; in this case the RDP between all pairs is 1, since they send to each other directly. However, the stress on links close to end hosts is high: Figure 1(b2) shows the paths taken by packets if A wants to communicate with all B, C, and D simultaneously. A must send three packets over its physical access link, one per overlay tunnel, leading to a stress of 3 on that physical link.

As another example, consider an overlay topology that selects overlay links randomly, resulting perhaps in the overlay shown in Figure 1(c1). In this case, all overlay tunnels go over the physical link R1-R2, leading to high stress. Additionally, the RDP between most pairs of nodes is very high (37/13 for (A,C), 24/2 for (A,B) and 25/2 for (C,D)). In general, topology agnostic schemes like CAN, Chord and Gnutella can yield such overlays.

Comparatively, a topology aware scheme (such as Narada) will result in an overlay topology that closely matches the underlying physical topology, as shown in Figure 1(d1). This topology has low RDP's between all pairs, and also results in low stresses on all physical links.

2.2 The Scalability of Overlays

To form a topology aware overlay, schemes like Narada introduce two sources of overhead: probe traffic between overlay node pairs, and the management overhead induced by membership changes and evolving network conditions. At a high level, we measure the scalability of an overlay scheme in terms of two metrics: the amount of state that is kept by each node to maintain the overlay, and the amount of network traffic that is attributable to overlay maintenance or network probing.

3 Kudos: a Scalable, Topology Aware Overlay

The management overhead associated with topology aware protocols is known to be prohibitive at scale. A simple and effective way to combat this overhead is to introduce hierarchy into the overlay. Kudos, our hierarchical overlay scheme, partitions overlay nodes into clusters. Each cluster has a unique representative node, called the cluster *head*; all non-head nodes in a cluster are referred to as *children*. In Kudos, we form two levels of overlays (Figure 2). At the bottom level, we run independent instances of a topology aware protocol within each cluster, forming an overlay of children for each cluster. At the top level, we run another instance of the protocol across cluster heads. In a Kudos overlay of n nodes, we form approximately $\sqrt{n}$ clusters, each consisting of $\sqrt{n}$ nodes (all of which are children, except a single cluster head). This hierarchy serves two purposes: it increases scalability since measurement probes are run across smaller groups, and it decreases management overhead by localizing the effects of member failures within smaller groups. However, a hierarchy potentially loses some opportunity for efficiency, since child nodes in different clusters do not have the ability to form overlay links to each other.

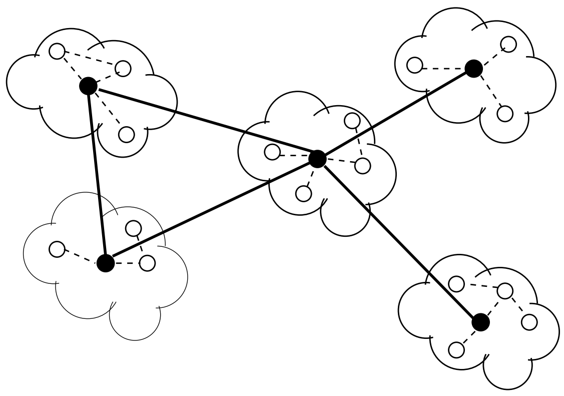

Figure 2: The Two-level hierarchy: clouds represent clusters with the solid nodes as head nodes. The solid lines between the heads is the top level topology.

The use of a two-level hierarchy divides the overlay construction task into two independent sub-problems: clustering and mesh management. Mesh management determines how the nodes at the same level are connected to each other. It comes into play at both levels of hierarchy; it determines how the nodes in the same cluster are connected to each other, and how the head nodes connect to each other (also called the *top-level* topology). Clustering deals with forming the clusters, selecting heads and maintaining them in the face of dynamic membership and changing network metrics. A node joins the overlay by being member of any cluster using some bootstrapping mechanism (for example, see [11]). With time, clustering operations will move the node to a more suitable cluster.

3.1 Mesh Management

The mesh management protocol selects the tunnels that connect the nodes at the same level. In Kudos, clustering operations are independent of mesh management and any suitable mesh-management protocol can be used. We use Narada [5] as our mesh management protocol. In Narada, participating nodes add links to each other, subject to the constraints of a maximum *degree bound.* A *utility function*[1] [5, 14] is computed periodically for each link in the overlay. The computed utilities dictate the replacement of existing links with poor utility with new links of better utility, thereby maintaining a good overlay in the face of dynamic conditions.

Each node also participates in a path vector routing protocol by periodically exchanging routing tables with its neighbors. Additional mechanisms are incorporated to maintain the connectivity of overlay in face of sudden node death. For full details on Narada, please refer to [5].

3.2 Clustering

There are two primary objectives of clustering: first, clustering attempts to assign nodes to clusters based on latency as a measure of proximity in the physical network, and secondly, clustering shuffles nodes across clusters to preserve the approximate correct size of each cluster. Cluster maintenance works through three clustering operations (*migrate*, *split*, and *diffuse*), which we describe below.

Migrate: This operation moves a child node from one cluster to another. Migrate tries to place a node in the cluster whose head is closest to that node. When a child joins the cluster, it receives information about other cluster heads from its current cluster head. (Each cluster head periodically broadcasts the list of other heads to its children using the bottom level overlay). The child then periodically probes a small number of carefully selected heads to determine its latency to them. Once the child detects a head node which is significantly closer to it than its current head, it leaves its current cluster and joins the new cluster. To avoid multiple migrations immediately after a node joins the overlay (as the initial joining

[1]Intuitively, the *utility* of having an overlay link is the extent to which the presence of that link improves its nodes' latency/bandwidth to other nodes.

cluster is picked randomly), a new node does not migrate until it has measured its latency to a significant fraction of other head nodes.

Migrate requires child nodes to probe all head nodes, which can overload head nodes in a large overlay. Heuristics can be used to reduce these probes. For example, in practice a child does not need to probe a head node that is more than $2 \times \mu$ from the child's head, where μ is the latency between the child and its head. In our experiments, this heuristic reduces the number of probes sent by up to 50%. Another possibility is to use *landmark ordering* [1] to choose appropriate cluster. In landmark ordering, every node (both child and head) measures its latency to a predefined set of landmark nodes and orders them. A child joins the head whose ordering most closely matches its own.

When a child node migrates to a new cluster, all the previous mesh management tunnels are deleted for that child. After joining the new cluster, the new head node randomly assigns it a neighbor to bootstrap the lower level mesh management.

Split: A cluster that is more than twice as large as $\sqrt{n}$ in size is broken into two. The decision to split a cluster is taken by its head node. After a split, the old cluster head remains as the head of one of the new clusters, but a new head must be selected for the second new cluster. The new cluster head is chosen to minimize the average latency to all other child nodes in its cluster; this decision requires knowledge of the latency between all pairs of child nodes.

Fortunately, every child node knows its latency to all other child nodes as a side-effect of the mesh management protocol. Whenever a decision to split occurs, the old cluster head broadcasts its latency to its child nodes. On receiving this information, each child node computes the number of other child nodes that are closer to than the old cluster head, and each child reports this number back to the old cluster head. After receiving enough reports, the old cluster head selects the new cluster head. The new head creates a tunnel to the old head, and from there on mesh-management enables it to get an appropriate set of tunnels to other cluster heads. The child nodes closer to the new head are informed of the split and they migrate to the new cluster.

Head selection in our implementation is based upon latency, however one can imagine incorporating other criteria, such as stability or access link bandwidth.

Diffuse: Over time, clusters may diminish in size because of node migrations and node deaths. Clusters that are more than a factor of two smaller than $\sqrt{n}$ are disbanded altogether, i.e., are diffused. All of the nodes in a cluster undergoing diffusion are migrated to neighboring clusters. To avoid the possibility of partitioning the top-level topology when the head node of the diffusing cluster migrates, links from its new head are made to all of its previous neighbors. Unnecessary links are eventually deleted by mesh management.

4 Topology Agnostic Approaches

To evaluate the performance of Kudos, we compare it to three topology agnostic overlay construction protocols. Our first two points of comparison, CAN [1] and Chord [2], are overlays used as the basis of a distributed hash table. Our third point of comparison is an overlay that reflects the power-law driven node degree distributions present in random overlays such as Gnutella [15, 16].

4.1 Content Addressable Network (CAN)

In CAN, nodes are mapped onto a virtual d dimensional Cartesian coordinate space. Every node has $2 \times d$ neighbors in the overlay, corresponding to its neighbors in the coordinate space. Since nodes are mapped randomly onto the Cartesian space, the overlay structure has no resemblance to underlying physical topology. We evaluate the following three variants of CAN overlays:

Naive. This is the original routing algorithm proposed in [1]; routes correspond to straight line paths through the Cartesian space the from a source coordinate to a destination coordinate. Next hop links are picked arbitrarily among those neighbors that are closer to the destination in the cartesian space, and as a result, many different path exists between two nodes in the space.

Smart. As suggested in [1], the naive routing algorithm can be enhanced; instead of arbitrarily picking next hops, a next hop neighbor is selected which is closest in the underlying physical network. Since higher dimensional CANs have more paths between any two nodes, the effect of this heuristic is more pronounced in higher dimensions.

Best (shortest path). In this variant, we run weighted shortest path calculations, with overlay link weights proportional to the latency across that link. This results in the best achievable performance on across the overlay structure. We study this variant for theoretical purposes only, since it is obviously impractical to use.

4.2 Chord

In Chord, every node is assigned a m bit identifier. For simplicity of explanation, we assume $n = 2^m$, where n is the number of nodes in the overlay. Every node has m neighbors (i.e. $m = log(n)$, and the number of neighbors grows with the overlay size), spaced by distance $2^0, 2^1, 2^2 \ldots 2^{m-1}$. The neighbors of node i are thus the nodes with ids $i + 2^0, i + 2^1, \ldots i + 2^{m-1}$. We evaluate three variants of routing across a Chord-style overlay:

Naive. While routing a packet from node i to node k, the packet is forwarded to the neighbor which is arithmetically closest to k, but less than or equal to k. For example, if node 0 routes to node 7, the next hop is 4. This is done successively, until the destination is reached. The above process ensures a route with maximum length $log(n)$ between any two nodes.

Smart. The routing algorithm can be enhanced by applying a heuristic mentioned in [17]. Intuitively, latency is reduced by preferentially choosing the next hop from a *suitable-set* of nodes, which is nearby in the underlying network.

Best or shortest path. This is the shortest path routing algorithm running over the overlay. Again, this is of theoretical interest only.

4.3 Random Power Law

In a random power law overlay, overlay nodes are connected randomly such that the node degree distribution obeys a power law. To generate a random power law overlay topology for n nodes, we first use the Brite [18] topology generator to generate a network structure with n nodes whose degree distribution obeys a power law. Overlay nodes are then mapped randomly onto this network structure. On a random overlay, we study two routing variants:

Flooding. Flooding has the advantage of being simple, but comes at the cost redundant packets in the network.

Best or shortest path. This is the shortest path routing algorithm running over the overlay. Once again, this is of theoretical interest only.

5 Experimental Methodology

To evaluate Kudos and to compare it to the traffic unaware algorithms, we made use of a locally developed event-driven packet level simulator. The simulator models the propagation delay of physical links, but does not model congestion, packet losses, or queueing delays. We deliberately chose to use this simplified network model, so as to factor out interactions with congestion control and issues relating to network capacity provisioning.

All topology agnostic approaches are simulated using a centralized algorithm for overlay construction, and we carefully control the order in which nodes join the overlay, depending on the experiment. Our primary interest in this paper is to understand the static properties of the overlay structure formed by these algorithms, rather than the overlays' performance under dynamic conditions.

We used the Georgia Tech "Transit-Stub" model (GT-ITM [19]) to generate the physical network topologies used in our simulations, and we attached additional nodes to the generated stub nodes to represent hosts connected to lower level routers. Latencies to links in the physical topology are assigned by GT-ITM.

For each data point gathered on each experiment, we ran 9 different simulations, representing 3 different topologies and three different random seeds. For each topology, the size of the simulated physical backbone was 4,040 nodes, and the number of stub nodes was approximately 20,000.

5.1 Performance Metrics

As discussed in Section 2, we use RDP and stress as metric to gage the latency overhead and bandwidth efficiency of an overlay network. We make use of the following, more specific metrics:

90th percentile RDP: RDPs are computed for every node pair in the overlay. The 90th percentile RDP is the RDP seen by 90% of the pairs of nodes. As mentioned in [5], we found that 90th percentile RDP hides sensitivity to simulation parameters by effectively factoring out high RDPs which are associated with pairs of nodes having very small physical latency.

Worst Link Stress: Recall that link stress for a is defined as the number of overlay links going over a physical link in a multicast data-transmission tree. The "worst link stress" is the maximum value of stress across all physical links involved in a data transmission. In our experiments, we compute single-source multicast dissemination trees as the union of the links used in the unicast routing from that source to all other nodes in the overlay.[2]

In our results, we compute the worst link stress for every multicast dissemination tree (i.e., we calculate the worst link stress experience by each multicast source). Our graphs plot the median of the worst link stress across sources; this is approximately similar to plotting the worst link stress of a randomly chosen multicast dissemination tree.

6 Hierarchy and Topology Aware Overlays

In this section, we present our experimental results that explore the effects of hierarchy on topology-aware overlays. The results explore two questions: how does hierarchy affect RDP and stress, and how does hierarchy affect the convergence time of the overlay under dynamic membership changes? To answer these questions, we simulate both Kudos and Narada overlays ranging from 64 to 1024 nodes, using the methodology we described earlier. For the results presented in this section, we chose an average out-degree of 6 for nodes.

6.1 RDP

Figure 3 shows the 90th percentile RDP values for Kudos and Narada, for different overlays sizes. We observe that the difference in RDP for Narada and Kudos is small (less than 20%) for overlay sizes over 1000 nodes; specifically, the RDP of a 1024 node Kudos overlay is 3.5, while the RDP of a 1024 node Narada overlay is 3.1. Also, note that RDP tends to increase slowly with overlay size for both Kudos and Narada; for Kudos, RDP increases from 2.9 to 3.5 as the overlay size changes from 128 nodes to 1024 nodes.

We conclude that the effect of hierarchy on packet delay is small; a 20% difference in RDP is unlikely to be significant in practice. Nonetheless, the effects of hierarchy can be observed in a very small fraction of node pairs, corresponding to situations in which physically close child nodes land in different clusters.

[2]Note that computing such a tree in a distributed manner can be complex, and different overlay management algorithms may have different ways and overhead of computing it. For our purposes, knowledge of the tree is sufficient, as we do not measure the overhead of computing it.

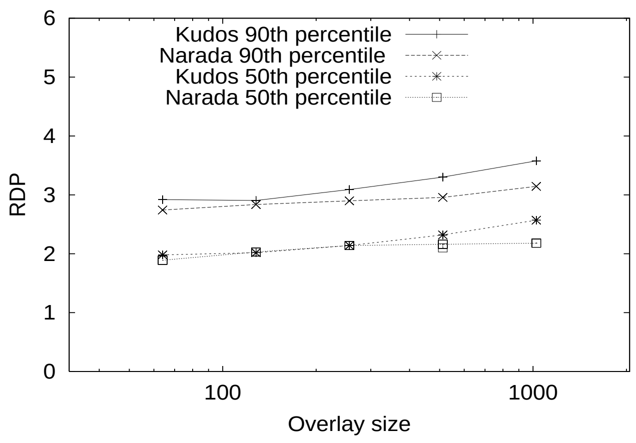

Figure 3: Comparing the 90th percentile RDP of Kudos and Narada, for various overlay sizes.

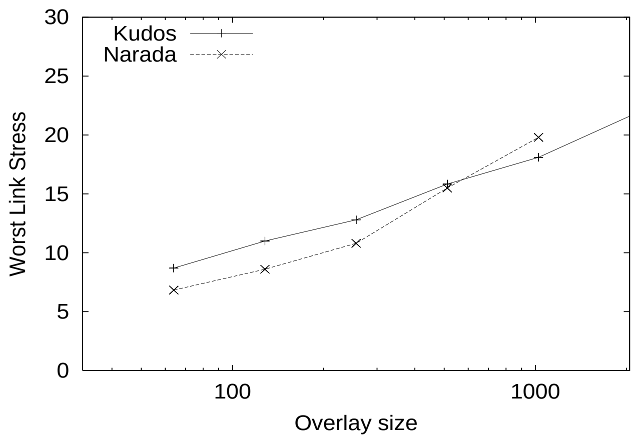

Figure 4: "Worst link stress" as measured for Narada and Kudos overlays of various sizes.

6.2 Link Stress

In Figure 4, we plot the relationship between worst link stress and overlay size, for both Narada and Kudos. The difference between the stresses across the algorithms is extremely small in practice, once again leading to the conclusion that hierarchy does lead to significant degradation in performance (in this case, in terms of bandwidth consumed).[3]

Worst link stress increases slowly for both approaches as a function of the overlay size; for Kudos, stress increases from 12 to 18 as the overlay size increases from 128 nodes to 1024 nodes.

[3]We confirmed that our results are comparable to those in [5], at least for overlay size ranges that we compute in common.

# of node changes	Kudos	Narada
4	10	120
8	15	214
16	27	295
32	30	342
64	31	745
128	33	1182

Table 1: The convergence time, expressed in number of protocol rounds before the Kudos or Narada overlay stablizes, for various sizes of group membership change.

6.3 Convergence Under Dynamic Group Membership

To study the impact of hierarchy on the behavior of the overlay algorithms during periods of dynamic membership, we measured the time it took for Kudos and Narada to converge to a stable overlay topology, as measured by the number of overlay maintenance protocol rounds that were necessary.

In Table 1, we show the number of protocol rounds before the convergence of a 256 node overlay, as a function of the number of nodes that simultaneously change. A “change” is defined as one node leaving the overlay, and another node joining in a different location. Thus, a 16-node change occurs when 16 nodes leave the overlay, and 16 different nodes join the overlay.

The results are dramatic: the convergence time of Kudos grows approximately as the square root of the number of nodes, while the convergence time of Narada is linear in the number of nodes. This, of course, is a result of Kudos clustering nodes into many, smaller overlays. In contrast, all nodes in Narada participate in a single, large overlay.

7 The Effectiveness of CAN and Chord Routing Variants

This section evaluates the effectiveness of the CAN and Chord routing variants with respect to RDP and worst case stress. The “best” (shortest-path) routing variant sets a baseline of performance for RDP; as such it represents the best possible RDP achievable for that overlay. However, as we will show, the routing variants have surprising effects on stress.

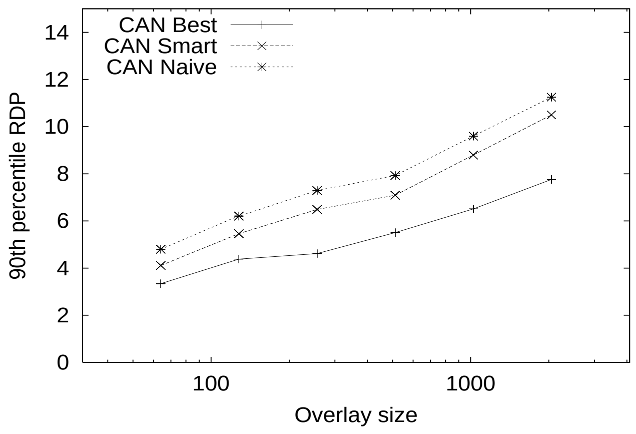

Figure 5: The performance of the three CAN routing variants, as measured by RDP on a 3-dimensional overlay of varying sizes.

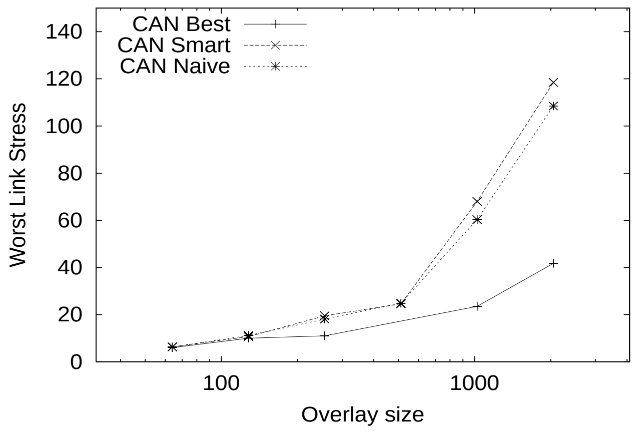

Figure 6: The performance of CAN routing variants, as measured by worst link stress.

7.1 CAN

In Figure 5, we plot the measured RDP of a 3-dimensional CAN; a 3-dimensional CAN has average node degree of 6, similar to our evaluated Kudos overlays. The graph shows the effect of overlay size on RDP, for all three routing variants.

As expected, the “best” routing scheme significant outperforms both “naive” and “smart” routing; the difference between naive and smart is much smaller. The number of neighbors of a CAN node is independent of the overlay size; as a result, the smart routing variant has fewer paths to explore, compared to best. Thus, the gap between best and smart routing grows as the overlay gets larger, relative to the gap between smart and naive.

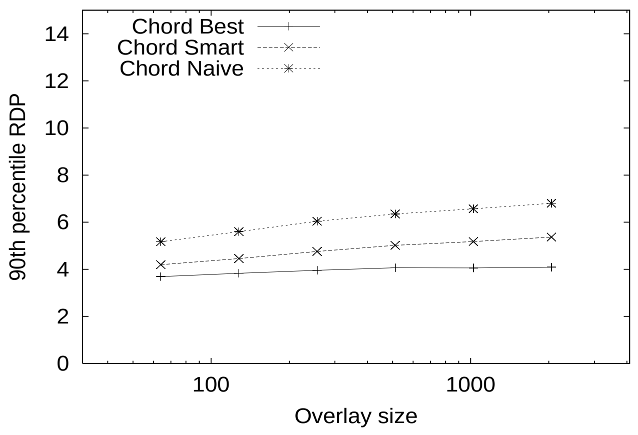

Figure 7: The performance of the Chord routing variants, as measured by RDP on overlays of varying sizes.

In Figure 6, we plot the worst link stress as a function of overlay size, again for the three CAN routing variants. For CAN, the best routing variant also has the smallest stress; naive and smart are essentially identical. Because best selects lower latency overlay links, it likely traverses fewer physical network links, resulting in lower worst link stress.

7.2 Chord

Figure 7 shows the 90th percentile RDP for the three Chord routing variants. Similar to CAN, best outperforms smart, and smart outperforms naive. Figure 8 shows the variation in link stress with the overlay size for different heuristics. Surprisingly, naive routing performs best, even though longer paths are taken as compared to smart. We believe this is because “naive” Chord routing perfectly diffuses the set of all point-to-point routes across overlay links, whereas CAN tends to favor certain overlay links. However, best and smart routing in CAN performs similarly to best and smart routing in Chord.

8 Comparing Kudos to Topology Agnostic Overlays

In this section, we present the results of comparing Kudos to CAN, Chord, and random power law overlays. In all experiments, the average node out-degree for Kudos, CAN and random power law overlays was set as 6. For CAN and Chord, we used the “smart” routing variant, whereas for random power law topologies, we used flooding-style routing.

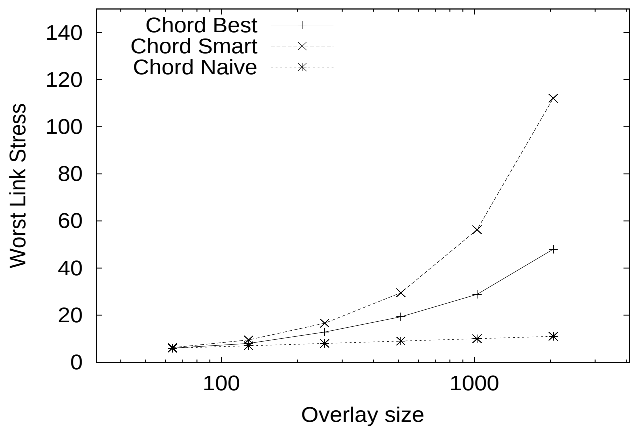

Figure 8: The performance of Chord routing variants, as measured by worst link stress.

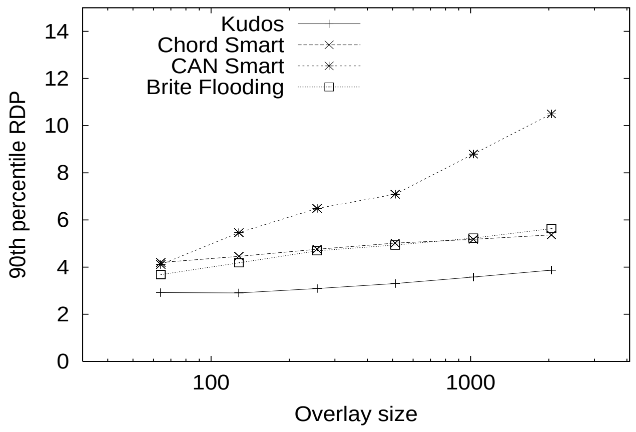

Figure 9: The 90th percentile RDP of the various overlay topologies as a function of overlay size.

8.1 RDP

Figure 9 shows the 90th percentile RDP values for these different protocols as a function of overlay size. A number of conclusions can be drawn. First, Kudos has much lower RDP than any other algorithm; hence, there is a clear advantage to using topology aware routing. Second, even though Chord appears to perform better than CAN, this is because in Chord, the average out-degree increases with overlay size.

8.2 Stress

In Figure 10, we show how stress is related to overlay size for our various different overlay algorithms. What is most apparent is that Kudos has much lower stress. Furthermore, the stress of Kudos grows much slower as a function of overlay size, as compared with all other algorithm. As before, this is a result of

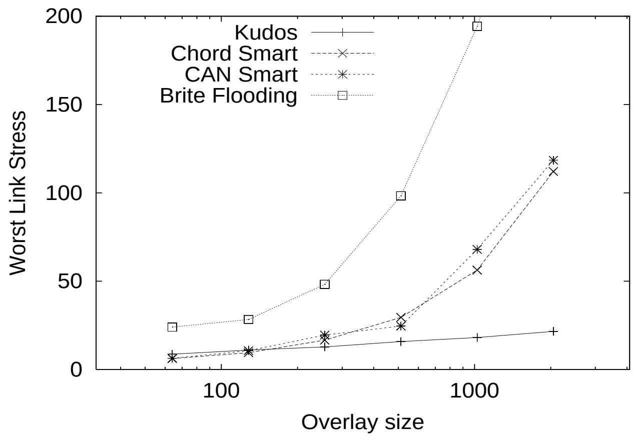

Figure 10: Worst link stress as as function of overlay size, for the various routing algorithms.

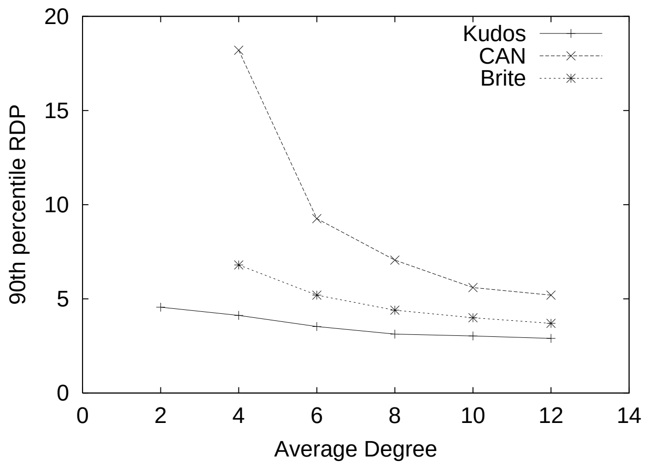

Figure 11: The effect of varying node outdegree on 90th percentile RDP.

Kudos' topological awareness, as overlay links tend to not traverse many physical network links. Random power law overlays have significantly worse stress than other algorithms, due to its use of flooding as its routing mechanism.

8.3 Effect of Increasing Degree

Next, we explored the effect of increasing the average out-degree of nodes in the various overlays. Increasing out-degree will decrease RDP, but at the cost of increasing stress. For the purposes of this section, we simulated a 1024 node CAN, Kudos, and random power law topology. We did not simulate Chord, since we could not control the out-degree of Chord nodes.

Figure 11 shows our results. As expected, there is a exponential reduction in RDP for CAN as the average node degree increases from 4 to 6, as this results in a corresponding increase in dimensionality.

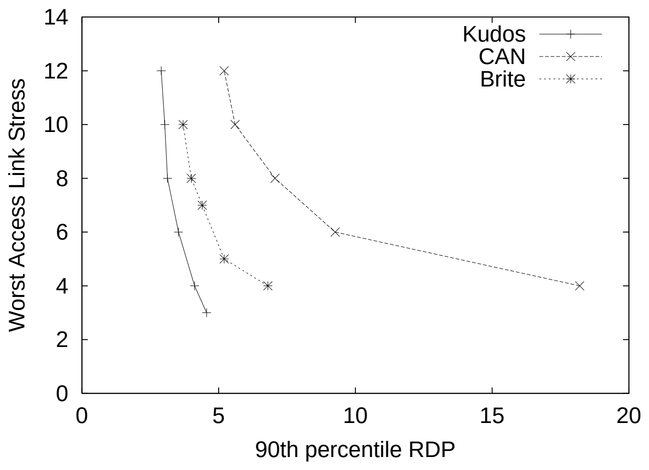

Figure 12: The tradeoff between RDP and access link stress as node out-degree changes, for both topology aware and topology agnostic overlays.

The same overall trend is observed for Kudos and random power law overlays, however, Kudos achieves excellent RDP performance with a much lower out-degree than its competitors.

In Figures 12 and 13, we explore the relationship between RDP and link stress, as a parametric function of node degree. Figure 12 shows this parametric relationship for access links (i.e., links between hosts and their routers in the physical network), whereas Figure 13 shows the same relationship for backbone links (i.e., physical links between routers).

All results show a fundamental tradeoff between RDP and link stress; by increasing average out-degree, an overlay has a smaller diameter, and hence needs to traverse fewer overlay links, and corresponding, physical links. However, because there are more overlay links, each physical link suffers from higher stress on average.

Both graphs confirm that topology-aware overlays achieve better overall performance than topology agnostic overlays: the Kudos parametric line lies closer to the origin than all others.

8.4 Overlay Management Overhead

Overlay management overhead is a measure of the amount of resources required by each node to construct and maintain the overlay. Nodes may need to maintain routing state, and nodes may need to exchange messages to perform overlay maintenance. For all protocols that we considered, state management and

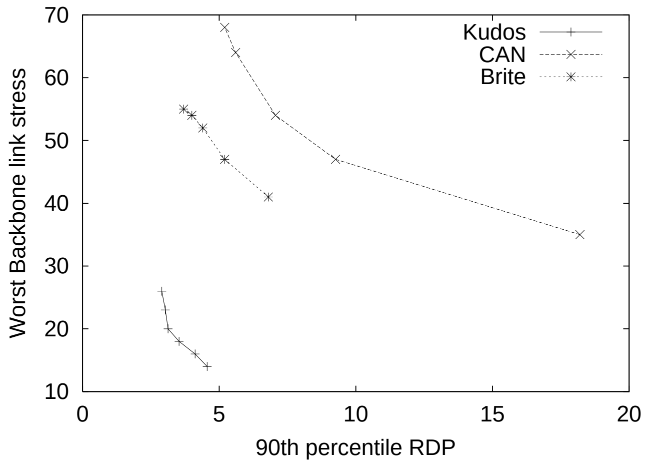

Figure 13: The tradeoff between RDP and backbone link stress as node out-degree changes, for both topology aware and topology agnostic overlays.

maintenance traffic grow identically as a function of the number of overlay nodes (or, for CAN, the dimension).

In Narada, every participant probes every other node in the overlay to choose suitable links; this causes each each node to exchange $O(n)$ messages. In *Kudos* , the mesh management protocol is run among groups of size $\sqrt{n}$, and hence overhead is $O(\sqrt{n})$. [4] Clustering operations are limited to communication among a cluster and hence are also $O(\sqrt{n})$. For CAN and Chord, the overlay maintenance communication is limited to neighboring nodes. Hence, for CAN, maintenance complexity is $O(2 \times d)$ and for Chord it is $O(log(n))$. For random power law overlays, assuming flooding as the routing mechanism, the only cost is maintaining links to neighbor nodes.

Table 2 summarizes the overlay management overhead for various protocols. Clearly topology agnostic approaches like CAN, Chord and random power law can scale much further then topology aware approaches like Narada and Kudos, although Kudos can scale much further than Narada.

9 Related Work

While many overlay construction schemes have been proposed (for example, see [5, 1, 2, 6, 8, 9, 20, 12]), there has been little work in studying how they compare to each other. Little is known about the

[4]Head nodes participates in two mesh management protocols, but the complexity order remains same, since both are $O(\sqrt{n})$.

Protocol	Overlay Management Overhead (bandwidth)
Narada	n
Kudos	$\sqrt{n}$
CAN (dimension d)	$2 \times d$
Chord	$log(n)$
Random Power Law (average degree d)	d

Table 2: Overlay Management Overhead for various schemes. n is the total number of nodes in the overlay

fundamental tradeoff of one scheme over another. We view our work as being a first, important step in this direction.

In this paper, we described the design of a hierarchical, topology aware overlay. Similar efforts are underway in [21] and [22]; perhaps unsurprisingly, both of these also use hierarchy to scale. [21] builds large scale overlays for multicast using multiple levels of hierarchy; at each level, the topology is a fully connected mesh, unlike Kudos in which the topology is sparse and is created by another self-organizing protocol. We believe that the design choices made by Kudos reduces the stress on links close to cluster representatives substantially, as compared to [21].

In [22], the authors provide an analysis of latency and cost optimizations in overlays. It concludes that hierarchy can provide significant benefits in scalability, with little performance cost. However, it does not present a distributed algorithm to construct hierarchy, which Kudos does. Additionally, their comparison is limited to topology aware algorithm, and no scalable topology agnostic scheme is compared.

10 Conclusions and Future Work

This paper has made two contributions: first, it presented the design of Kudos, a hierarchical extension to the Narada topology aware overlay construction algorithm. Using simulations, we demonstrate that Kudos has superior scalability than Narada while maintaining the performance advantages of topology aware overlays. Because of its use of hierarchy, Kudos also has lower management overhead than Narada.

Our second contribution was a detailed, quantitative comparison between topology aware and topology agnostic overlay algorithms. Specifically, we compared Kudos to CAN, Chord, and a random power law overlay topology. We demonstrated that all overlay topologies are faced with a fundamental tradeoff between relative delay penalty (RDP) and link stress, and that by changing node out-degree, different points in this tradeoff can be selected. We further demonstrated that Kudos significant outperforms all considered topology agnostic algorithms, although it cannot scale to the same degree because of the costs incurred by topology aware overlay maintenance.

In the future, we hope to extend our comparisons to approaches like *Tapestry and Pastry*, which are scalable and at the same time can more easily create topology aware overlays. We also hope to explore the performance of Kudos under more dynamic environments, such as rapid overlay membership changes, or varying traffic demands.

References

[1] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker, "A Scalable Content Addressable Network," in *ACM SIGCOMM*, August 2001.

[2] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan, "Chord: A peer-to-peer lookup service for internet applications," in *ACM SIGCOMM*, August 2001.

[3] H. Eriksson, "MBone: The Multicast Backbone," in *Communications of the ACM*, August 1994, vol. 37, pp. 54–60.

[4] "6Bone Web Pages," http://www.6bone.net/.

[5] Yang-hua Chu, Sanjay Rao, and Hui Zhang, "A Case for End System Multicast," in *ACM SIGMETRICS*, June 2000.

[6] Yatin Chawathe, S. McCanne, and Eric Brewer, "An Architecture for Internet Content Distribution as an Infrastructure Service," February 2000.

[7] "Gnutella," http://www.gnutella.com.

[8] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph, "Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing," Tech. Rep. UCB/CSD-01-1141, UCB, April 2001.

[9] Anthony Rowstron and Peter Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, November 2001.

[10] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris, "The Case for Resilient Overlay Networks," in *HotOS VIII*, May 2001.

[11] Yatin Chawathe, Steven McCanne, and Eric A. Brewer, "RMX: Reliable Multicast for Heterogeneous Networks," in *INFOCOM (2)*, 2000, pp. 795–804.

[12] John Jannoti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole, "Overcast: Reliable Multicasting with an Overlay Network," in *OSDI*, 2000.

[13] Dimitrios Pendarakis Tellium, "ALMI: An Application Level Multicast Infrastructure," .

[14] Yang hua Chu, Sanjay G.Rao, Srinivasan Seshan, and Hui Zhang, "Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture," in *ACM SIGCOMM*, August 2001.

[15] Stefan Saroiu, P. Krishna Gummadi, and Steven D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," in *Multimedia Computing and Networking*, 2002.

[16] DSS2 Clip System, "Gnutella: To the Bandwidth Barrier and Beyond," "http://dss.clip2.com/gnutella.html".

[17] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica, " Wide-area cooperative storage with CFS ," in *ACM SOSP*, October 2001.

[18] Alberto Medina, Ibrahim Matta, and John Byers, "On the origin of power laws in internet topologies," Tech. Rep. 2000-004, 20, 2000.

[19] "GT-ITM: Modeling Topology of Large Internetworks," http://www.cc.gatech.edu/projects/gtitm/.

[20] Paul Francis, "Yoid: Your Own Internet Distribution," http://www.aciri.org/yoid/.

[21] Suman Banerjee, Bobby Bhattacharjee, and Srinivasan Parthasarathy, "A Protocol for Scalable Application Layer Multicast," Tech. Rep., University of Maryland, July 2001.

[22] Dejan Kostic and Amin Vahdat, "Latency versus Cost Optimizations in Hierarchical Overlay Networks," Tech. Rep., Duke University, June 2001.